Lab Exercise 6: Terraform Multiple tfvars **Files**

Objective:

Learn how to utilize multiple tfvars files in Terraform to manage configurations for different environments.

Prerequisites:

- Terraform installed on your system.
- Basic understanding of Terraform configurations and variables.

Steps:

1. Create a Terraform Directory:

Bash

mkdir terraform-multiple-tfvars

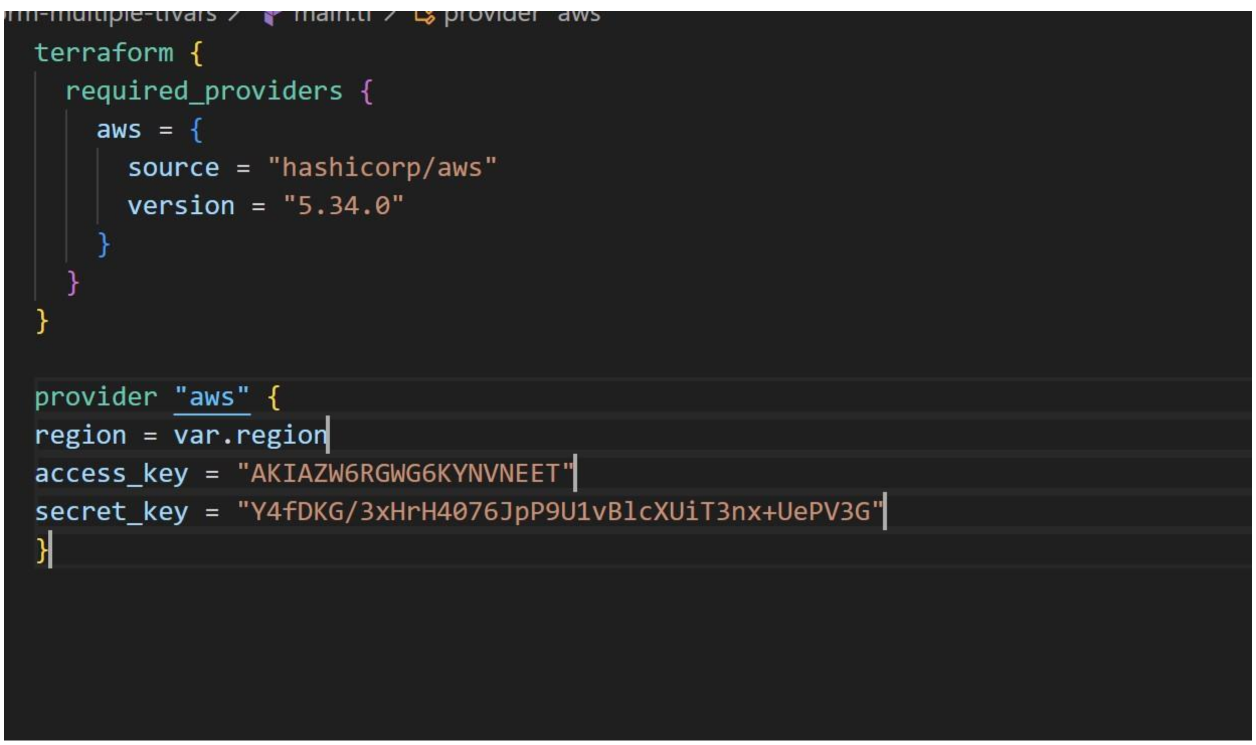
cd terraform-multiple-tfvars

2. Create Terraform Configuration Files:

a. Create a file named main.tf**:**

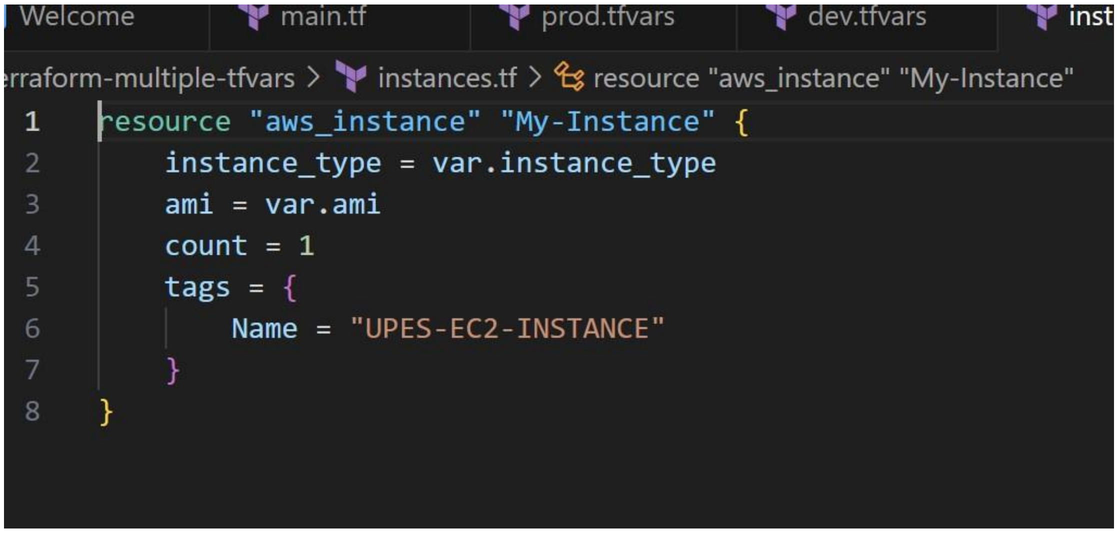
Terraform

main.tf



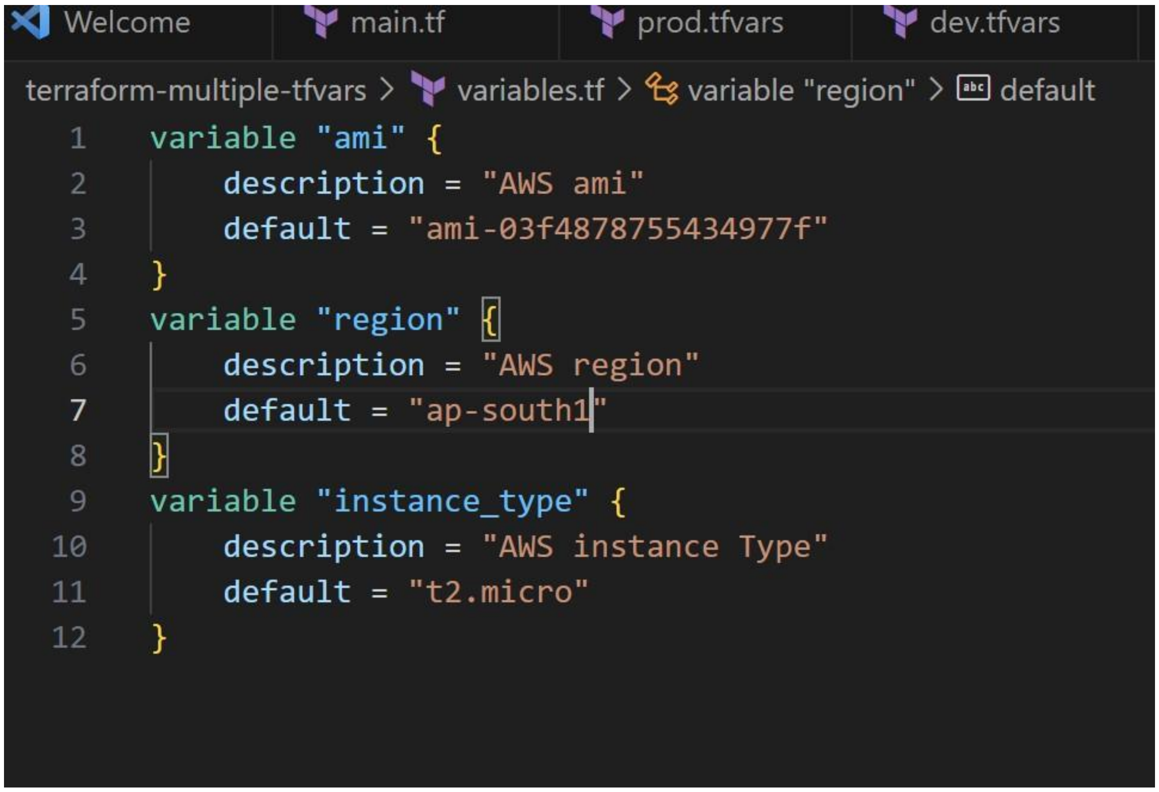
```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.34.0"
    }
  }
}

provider "aws" {
region = var.region
access_key = "AKIAZW6RGWG6KYNVNEET"
secret_key = "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"
}
```

b. Create a file named variables.tf**:**


Welcome
main.tf
prod.tfvars
dev.tfvars
inst
rraform-multiple-tfvars > instances.tf > resource "aws_instance" "My-Instance"

```
resource "aws_instance" "My-Instance" {
    instance_type = var.instance_type
    ami = var.ami
    count = 1
    tags = {
        Name = "UPES-EC2-INSTANCE"
    }
}
```

Vairable.tf

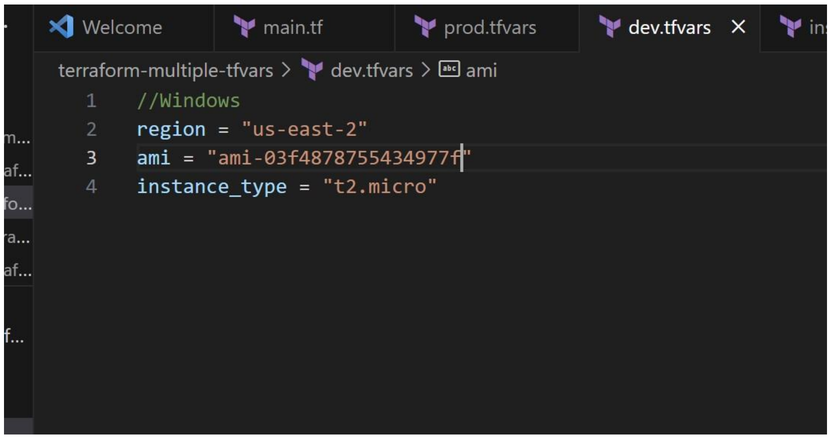

Welcome
main.tf
prod.tfvars
dev.tfvars
terraform-multiple-tfvars > variables.tf > variable "region" > default

```
variable "ami" {
    description = "AWS ami"
    default = "ami-03f4878755434977f"
}
variable "region" {
    description = "AWS region"
    default = "ap-south1"
}
variable "instance_type" {
    description = "AWS instance Type"
    default = "t2.micro"
}
```

3. Create Multiple tfvars **Files:**

a. Create a file named dev.tfvars**:**

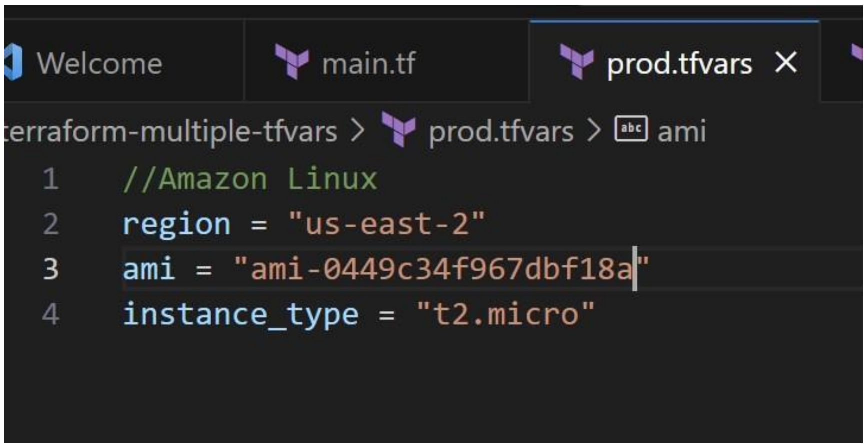
# dev.tfvars



```
//Windows
region = "us-east-2"
ami = "ami-03f4878755434977f"
instance_type = "t2.micro"
```

b. Create a file named prod.tfvars**:**

# prod.tfvars



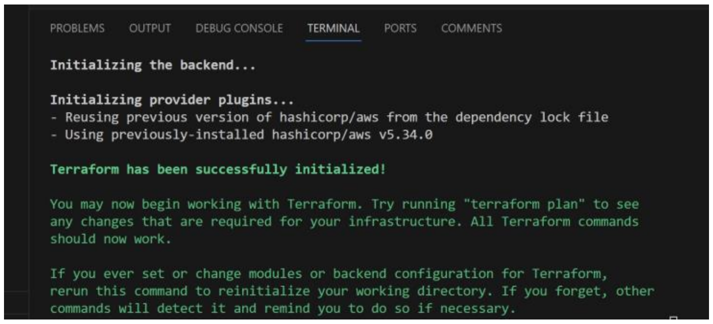
```
//Amazon Linux
region = "us-east-2"
ami = "ami-0449c34f967dbf18a"
instance_type = "t2.micro"
```

"

Explanation:

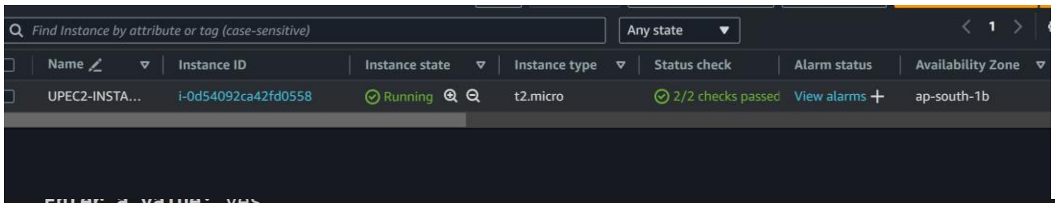
In these files, you provide values for the variables based on the specific environments (dev and prod).

4. Initialize and Apply for Dev Environment:

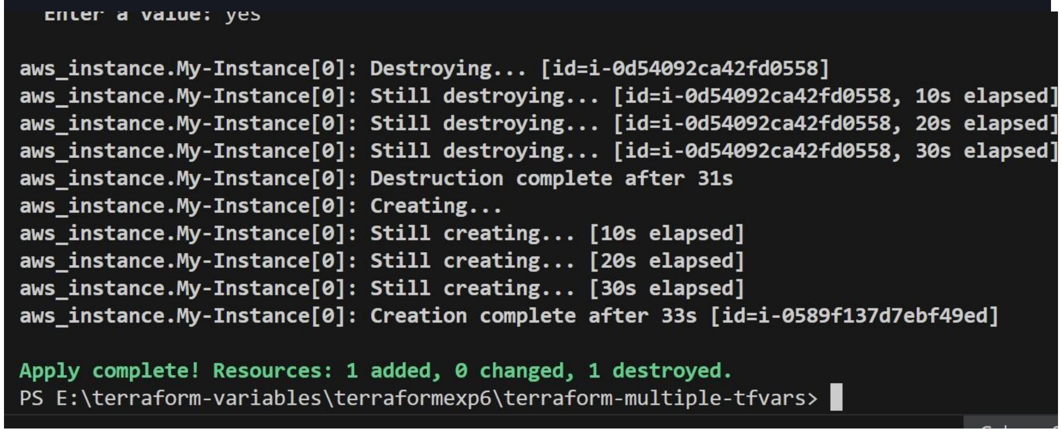


```
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.34.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Explanation:

- The terraform init command initializes the Terraform configuration and downloads the necessary AWS provider plugins.
- The terraform apply -var-file=dev.tfvars command creates the infrastructure for the development environment using the values specified in the dev.tfvars file.
- 

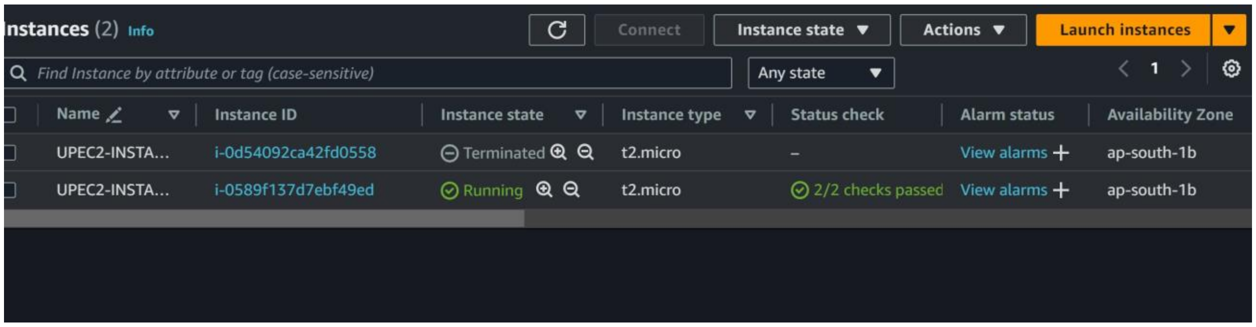
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
UPEC2-INSTA...	i-0d54092ca42fd0558	Running	t2.micro	2/2 checks passed	View alarms	ap-south-1b

- 

```
  Enter a value: yes

aws_instance.My-Instance[0]: Destroying... [id=i-0d54092ca42fd0558]
aws_instance.My-Instance[0]: Still destroying... [id=i-0d54092ca42fd0558, 10s elapsed]
aws_instance.My-Instance[0]: Still destroying... [id=i-0d54092ca42fd0558, 20s elapsed]
aws_instance.My-Instance[0]: Still destroying... [id=i-0d54092ca42fd0558, 30s elapsed]
aws_instance.My-Instance[0]: Destruction complete after 31s
aws_instance.My-Instance[0]: Creating...
aws_instance.My-Instance[0]: Still creating... [10s elapsed]
aws_instance.My-Instance[0]: Still creating... [20s elapsed]
aws_instance.My-Instance[0]: Still creating... [30s elapsed]
aws_instance.My-Instance[0]: Creation complete after 33s [id=i-0589f137d7ebf49ed]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
PS E:\terraform-variables\terraformexp6\terraform-multiple-tfvars>
```

6. Test and Verify:

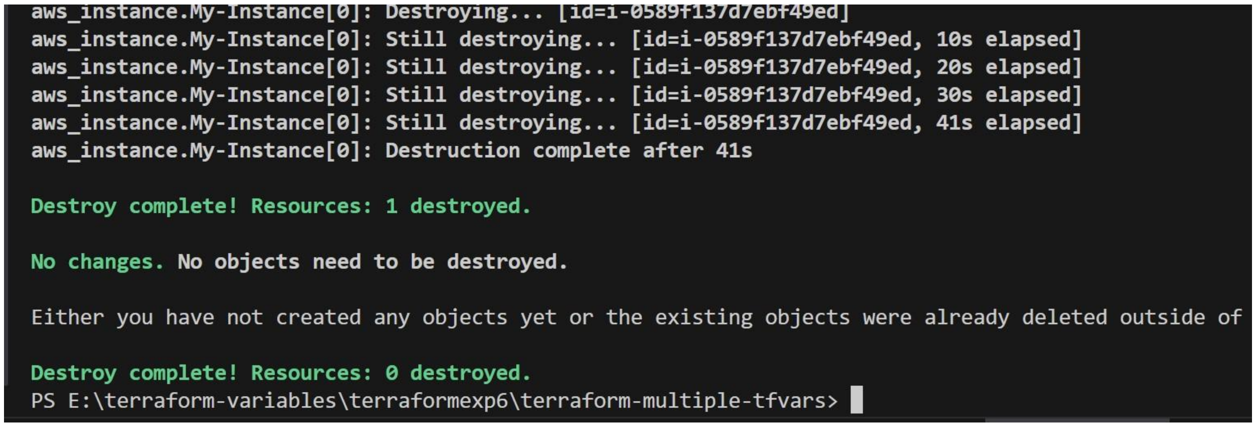
- Observe how different tfvars files are used to set variable values during the apply process for each environment.
- You can access the AWS Management Console or use the AWS CLI to verify the creation of resources in the specified regions and with the defined instance types.

- 

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
UPEC2-INSTA...	i-0d54092ca42fd0558	Terminated	t2.micro	–	View alarms +	ap-south-1b
UPEC2-INSTA...	i-0589f137d7ebf49ed	Running	t2.micro	2/2 checks passed	View alarms +	ap-south-1b

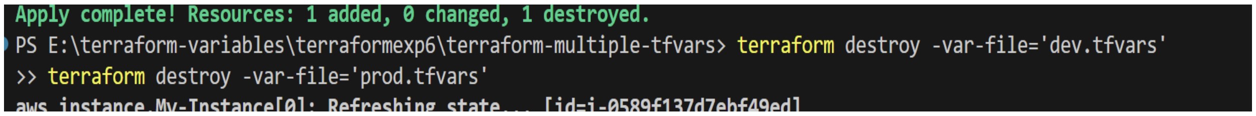
7. Clean Up:

a. Destroy Resources for Dev Environment:



```
aws_instance.My-Instance[0]: Destroying... [id=i-0589f137d7ebf49ed]
aws_instance.My-Instance[0]: Still destroying... [id=i-0589f137d7ebf49ed, 10s elapsed]
aws_instance.My-Instance[0]: Still destroying... [id=i-0589f137d7ebf49ed, 20s elapsed]
aws_instance.My-Instance[0]: Still destroying... [id=i-0589f137d7ebf49ed, 30s elapsed]
aws_instance.My-Instance[0]: Still destroying... [id=i-0589f137d7ebf49ed, 41s elapsed]
aws_instance.My-Instance[0]: Destruction complete after 41s

Destroy complete! Resources: 1 destroyed.

No changes. No objects need to be destroyed.

Either you have not created any objects yet or the existing objects were already deleted outside of

Destroy complete! Resources: 0 destroyed.
PS E:\terraform-variables\terraformexp6\terraform-multiple-tfvars>
```

b. Destroy Resources for Prod Environment:



```
Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
PS E:\terraform-variables\terraformexp6\terraform-multiple-tfvars> terraform destroy -var-file='dev.tfvars'
>> terraform destroy -var-file='prod.tfvars'
aws_instance.My-Instance[0]: Refreshing state... [id=i-0589f137d7ebf49ed]
```

Explanation:

- These commands destroy the infrastructure created for each environment respectively.
- Confirm the destruction by typing yes when prompted.

8. Conclusion:

This lab exercise demonstrates how to effectively leverage multiple tfvars files in Terraform to manage variable values for different environments. This approach allows you to maintain separate configuration files for each environment, simplifying infrastructure code management and maintenance. Feel free to experiment with different values in the dev.tfvars and prod.tfvars files to see how they influence the infrastructure provisioning process for each environment.